2019

Speech Enabled Navigation in Virtual Environments

Raksha Rajashekar
Wright State University

Speech Enabled Navigation in Virtual Environments

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

RAKSHA RAJASHEKAR
B.E.C.S, Visvesvaraya Technological University, 2015

2019
WRIGHT STATE UNIVERSITY

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

July 29, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Raksha Rajashekar ENTITLED Speech Enabled Navigation in Virtual Environments BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Computer Engineering.

Thomas Wischgoll, Ph.D.
Thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on
Final Examination

Thomas Wischgoll, Ph.D.

John Gallagher, Ph.D.

Yong Pei, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Rajashekar, Raksha. M.S.C.E, Department of Computer Science and Engineering, WRIGHT STATE UNIVERSITY, 2019. *Speech Enabled Navigation in Virtual Environments* .

Navigating in a Virtual Environment with traditional input devices such as mouse, joysticks and keyboards provide limited maneuverability and is also time consuming. While working in a virtual environment, changing parameters to obtain the desired visualization requires time to achieve by manually entering parameter values in an algorithm to test outcomes. The following thesis presents an alternate user interface to reduce user efforts, while navigating within the Virtual Environment. The user interface is an Android application which is designed to accommodate spoken commands. This Speech Enabled User Interface termed as the Speech Navigation Application (SNA), provides the user with an option to voice out the commands which they wish to see enacted/reciprocated in a virtual environment. The user can change the parameters to meet their needs. The idea behind this project was to minimize the effort needed to change parameters of any visualization, so as to obtain the desired view as per the requirement. This paper explains in detail the design, implementation and evaluation of the working project. This system is analyzed by simulating the working prototype in the DIVE environment [15].

Contents

List of Figures

Acknowledgment

I would like to take this opportunity to thank my Advisor Dr. Thomas Wischgoll, for his guidance and encouragement in helping me make this project successful. I would also like to thank Dr. Yong Pei and Dr. John Gallagher for their support with this project.

Last but not least, I would like to thank my parents Rajashekar Havaldar and Deepalaxmi Rajashekar for their constant support and encouragement in pursuing my dreams. I would also like to thank my siblings Deeraj Rajashekar and Darshitha Rajashekar, for inspiring me to be as creative as they are.

Background

Over the past couple of years, virtual environments have been used in many applications, ranging from airplane cockpit simulation [11] to virtual reality games. They have been used in multiple fields such as astronomy, medicine and data handling, providing a new platform for educational and training purposes. This trend has been popular among users. However, navigating in a virtual environment can sometimes become tricky. Be it when you are playing an interactive games or just for visualizing data. Navigating using joysticks or keyboards make fully immersive environments awkward to use with such bulky input devices. In case of Data visualization, the user may want to change the view multiple times to understand the data better. Constantly updating the algorithms manually and running them is time consuming and requires that you exit the current simulation and run it again after changing the parameters..

Many Virtual reality (VR) interfaces use traditional input devices like keyboards, handheld devices such as joystick and mouse, which offer limited maneuverability in the VR environment. Since certain VR environments require synchronous use of one or more device to get a desired view, this sort of multitasking, can often be confusing to users and time consuming. Hence more user-friendly approaches have been in demand to make user interaction with the whole VR experience more comfortable.

Virtual Environments have been proven to be useful in multiple instances. They have been used in fields such as manufacturing, agent and user interaction, simulations and industrial application [17]. The paper that involves the use of VR Environment in Industrial

application, gives an instance of speech used in a virtual immersive environment. Here they have used python programming in Jack python console, which serves as the client for voice command output by the Resin Server. They have explained the use of voice commands to create the interactive process between operators, tools and components. It also explains about how using spoken commands in immersive environments to simulate interaction between operators and tools as a digital representation instead of developing physical prototypes, are cost efficient and user friendly.

Another scenario involves the Virtual Shopping Environment [3]. This implementation is used to simulate a shopping environment that mimics the actual shopping experience. The situation however is when the user knows the particular item that they want, getting to that isle without having to manually go through each isle is the task. Using a command to navigate to that isle or mentioning the name of the item that they wish to get, can be beneficial.

My focus for the thesis lies in the field of Data Visualization and analysis. For the most part, the VR environments provide ideal displays to visualize data in depth as it provides a more interactive environment to view data up close in large displays. The traditional user interfaces that are used for such simulations are keyboards, joysticks and mouse.

To provide a more comfortable and familiar experience to the VR environment, a more readily available and easily adaptable input is required. Considering that most people own smartphones and are familiar with the process of using applications on their smart-phones makes it a user-friendly and easy to use input device for a VR experience. Considering an increasing trend in VR experience, a readily available input device is preferred as it will be multi-functional and at the same time less expensive to own. An application that works with an environment, rather than buying a device that could be used in the environment would be a more cost-effective interface, which can be readily used by the public, in scenarios such as a Virtual Shopping environment. The equipment that is used can be expensive, an app however is something that is easily accessible to most people and is less expensive.

The sheer fact that many people are familiar with how an app functions makes it more user friendly. This being considered, is a good investment if we were to see an increase in immersive experiences, like in the case of a virtual shopping environment [3]. The user will be required to navigate through the visualization to shop for the desired groceries. Hence, I devised the Speech navigation Application (SNA). The entire working system is termed as the Speech Navigation System (SNS) This application enables users to control the visualizations as per their needs through spoken commands. The applications of such an app can be useful in many scenarios, such as data visualization and changing parameters to alter the visualization.

A paper on ”Speech-enabled augmented reality supporting mobile industrial maintenance” [5], explains the use of a speech-enabled augmented reality framework which is used to interact with the factory components. An example that they have mentioned involves a communication similar to the implementation of the SNS, but in an industry scenario and the. Here, the user while walking by a component asks a question like ”whats the status?” and the augmented framework responds with the appropriate response with regards to the component as a result. The SNA however differs in result. It does not relay any speech response as a result, but an alteration in the visualization in the VR environment's display. Another such scenario where audio based interaction is used is in Virtual learning environments [4].In this paper they have introduced the idea of a ”Voice interactive Classroom” and provide a overview of how the audio based learning is helpful.

Even though speech-enabled communication gives a lot of freedom to the user in terms of communicating with an environment, there are quite a few constraints associated with this form of communication/input. The paper titled ”User Interface constraints for Immersive Virtual applications”[2], explains in detail about many of the issues or constraints of using various input methods for virtual immersive environments including Voice-Recognition. Here, they have mentioned the various advantages and constraints of Voice-Recognition systems. This is relevant as I have used Google Cloud Speech-to-Text API,

though it is very useful and coverts the user commands the user provides, it still has a few constraints. These constraints will be explained in more detail in the upcoming sections.

1.1 Technologies Used

In order to implement the Speech Navigation System, I had to use a combination of various technologies such as AWS, VRUI, VTK and Google Cloud Speech-to-text API. Each of these technologies have played an integral role in the implementation of the Speech Navigation System (SNS). I will provide a brief overview of each software that I have used, and their applications in the following sections.

1.1.1 Android studio

Android studio is an IDE which allows you to design applications that can be adaptable to run on different types of devices like tablets, phones and TVs. I used Android studio to develop the Speech Navigation Application (SNA). It supports the integration of a number of API's. The book that I referred to while working with the SNA is called "Starting with Android" by Dr. M. M. Sharma and Rashmi Aggarwal [13].

1.1.2 Amazon Web Services (AWS)

Amazon Web Services is a cloud computing platform that provides a variety of services. I have used AWS EC2 instance. Here, EC2 stands for Elastic compute cloud and EC2 instance is a virtual server in Amazon's EC2. It can be used to run applications on the AWS infrastructure. AWS provides users these services at a reasonable rate. There are also free tier services provided that includes 750 hours of the Linux and Windows t2.micro instances for each month for a year. The pros of using Amazon EC2 instance is that the cloud hosting provides good backup, and also doesn't require us to maintain local hardware

resources. The cons of this is that it requires good technical knowledge to set up and run the application. There are however many tutorials that are available to help set up your instance. I decided to work with EC2 instances as it is affordable and also manages your data.

1.1.3 Virtual Reality User Interface (VRUI)

VRUI stands for Virtual Realty User interface. It was designed by Oliver Kreylos [8] for high- performance VR applications. VRUI was designed to be a "completely environment-independent software". It has been used in various applications such as gaming and data analysis. VRUI supports a number of VR devices, and it masks the user from the integration requirements as it does so on its own. There are multiple applications of VRUI in various fields mainly for medical and gaming applications. VRUI can be used to add new devices without the need to make changes to the application. The VRUI has a tool layer that separates the VR Application from the input Devices.

VRUI has been used in the implementation of the Speech Navigation System(SNS). In [15], it shows how the VRUI is used to integrate all the individual displays that comprise the DIVE environment to display one large visualization. The same DIVE environment is used for visualizing the VTK algorithms for the SNS. VRUI toolkit takes the output form the VTK algorithms and displays it on the DIVE environment as one big display by handling the display dependencies.

1.1.4 Visualization Toolkit (VTK)

VTK stands for Visualization Tool Kit [12]. The Visualization Tool Kit was developed by Will Schroeder, Ken Martin, and Bill Lorensen. VTK can be used to create vivid visualizations and applications as well as VTK can be used to run complex simulations in VR environments. As it supports a vast number of libraries, it can process any form of

input data, provided you use the correct libraries. The VTK pipeline or code is setup using source, filters, mappers, actors, renderers and windows. The source represents the input data in the form of a file or generated data. The source is followed by the filters, they are used to modify the data to be visualized. These filters are then followed by mappers, which convert the data into objects. The mappers are followed by actors. Actors represent the data to be visualized and any alterations in terms of display can be made to the actor. for instance, changing the color of the data displayed etc. These actors are then followed by the Renderers and windows, the renderers are the viewports or the window in which the data to be visualized is displayed. It has been used for visualizing complex data in various fields, from industrial simulations to medical education/training [9].

One such instance where VTK has been used in the medical field, involves the visualization of three-dimensional ultrasound images [14]. Here, they have used VTK libraries to produce a three-dimensional reconstruction of fetal ultrasound using surface rendering.

In a paper published in 2007 on VTK [7] using multimodal virtual reality interface for three Dimensional (3D) interaction with VTK, focuses on the use of an alternative input method similar to the one that I am trying to implement. They mention that the user can control the commands given to the VTK environment through Speech. This paper is most relevant to the implementation that I am trying to achieve with the Speech Navigation System. The paper stated that they focused on the Speech based commands so that the user would have more focus on the visualization. This paper also stated that user had to learn some commands and those spoken commands weren't always interpreted accurately. Also, the users were not able to see or know whether the spoken commands were received.

1.1.5 Google Cloud Speech-to-Text API

The Google Cloud Speech to Text API is one of the AI and Machine Learning Products that Google provides. This API was essential to my project's implementation. Based on a study conducted on testing the efficiency of various available API's such as the Microsoft API,

Sphinx-4 and the Google API [6], the Google API fared better. According to the study, the Google API has a better acoustics and language model. This is one of the reasons I decided to go with this API. The Google Speech to Text API as the name suggests, converts the users spoken content to text.

Another advantage of the Google Cloud Speech-to-Text API is that it supports multiple languages, and can be incorporated to communicate with the VR environments using multiple languages [1]. This could be useful if the SNS system were to be used in various applications in different parts of the world by localities who might not be familiar with the English language.

1.1.6 Display Infrastructure for Virtual Environments (DIVE)

The paper "Display Systems for Visualization and simulation in virtual environments"[15], represents how the DIVE environment was designed to mimic similar functionality to that of a CAVE environment[10]. Here, the Dive environment is assembled using flat panel TV's. These TV's are arranged in a way to provide the display configuration with 270 degrees surrounding view, for immersion. By using the flat panel TV's, the resolution is considerably more, and the large displays provide more brightness.

The DIVE environment is used for the SNS to provide a large-scale visualization of the VTK algorithms. The VRUI toolkit was used to integrate the display with the DIVE environment. Figure 3.1 shows the incorporation of the DIVE environment with the Speech navigation System.

Introduction

Traditional input devices like keyboards, handheld devices such as joystick and mouse offer limited maneuverability in virtual environments. Certain visualizations require synchronous use of one or more devices to get to a desired view. This sort of multitasking can often be confusing to users and time consuming. Also, altering visualizations manually takes time to achieve, especially when you have multiple parameters you wish to alter. Hence, in recent days, a more user-friendly approach is preferred over traditional methods to navigate in Virtual Environments. Therefore, the Speech Navigation Application was developed. The system allows users to voice out the commands which they wish to see reciprocated a virtual environment.

Android studio and Java socket programming were used to design the application. I incorporated Google's speech to text API in the app, to convert the user's spoken commands to text. The server was implemented using Amazon Web Services. Specifically, I used the EC2 Ubuntu instance to program my server back-end. It is programmed using Python and I used python socket programming to establish communication between the server and its respective clients. The client side was implemented using OpenGL, Visualization Toolkit (VTK), and Virtual Reality User Interface (VRUI). The display system that we used to test the working code is the DIVE Environment. The DIVE environment [16]was used to test the working of the app in the virtual environment. The client side was programmed using C++ programming language and various VTK algorithms were used in the process of implementing this system. VRUI helped interface the VTK code and thedisplay requirements

for the system to be executed in the DIVE environment [15].

The system was tested by running the commands and making a note of the corresponding changes in the visualization. Several participants were asked to test the working system. Each participant was given a task to alter certain parameters in the visualization, using the application. The same participants were then asked to manually change the parameters. Based on their experience, they were asked to fill out a brief survey, which included questions that asked them to rate their experience based on usability and preference.

Implementation

The Implementation of this System can be divided into 3 main sections, namely Speech Navigation Application, Server design and implementation and Client-side design and implementation. Each section was designed individually and then combined to form the final working system that I would like to term as the Speech Navigation System(SNS). Figure 3.1represents the overall System architecture and the interactions between the Application, Server and Display Client.

The Application that I have developed also known as the Speech Navigation Application (SNA), is the alternate user interface that I have designed to replace the use of traditional input devices such as the keyboard, mouse, joystick etc. in navigating through the virtual environment. As majority of the current generation have access to a smart phone, an application is not just cost effective, but also more convenient as a majority of the users are familiar with how to use an application. This input method would be more convenient for users as a smartphone is multi-functional, they will not have to buy bulky input devices to work with the virtual environment which are not only expensive but also make it awkward to use in immersive virtual environments.

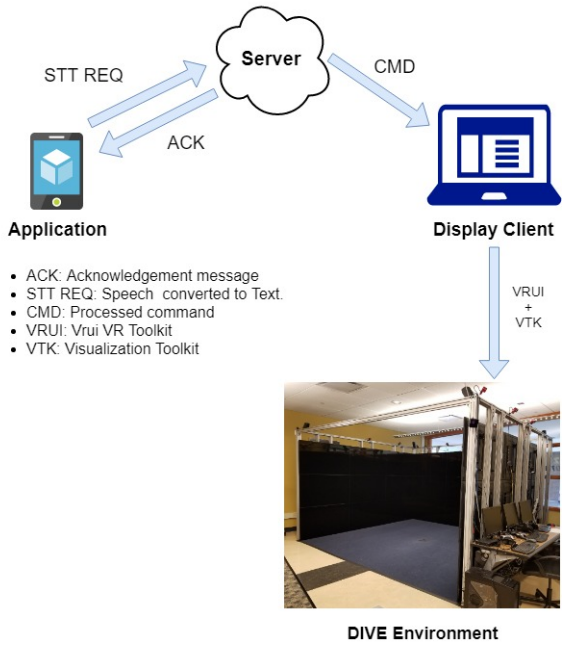


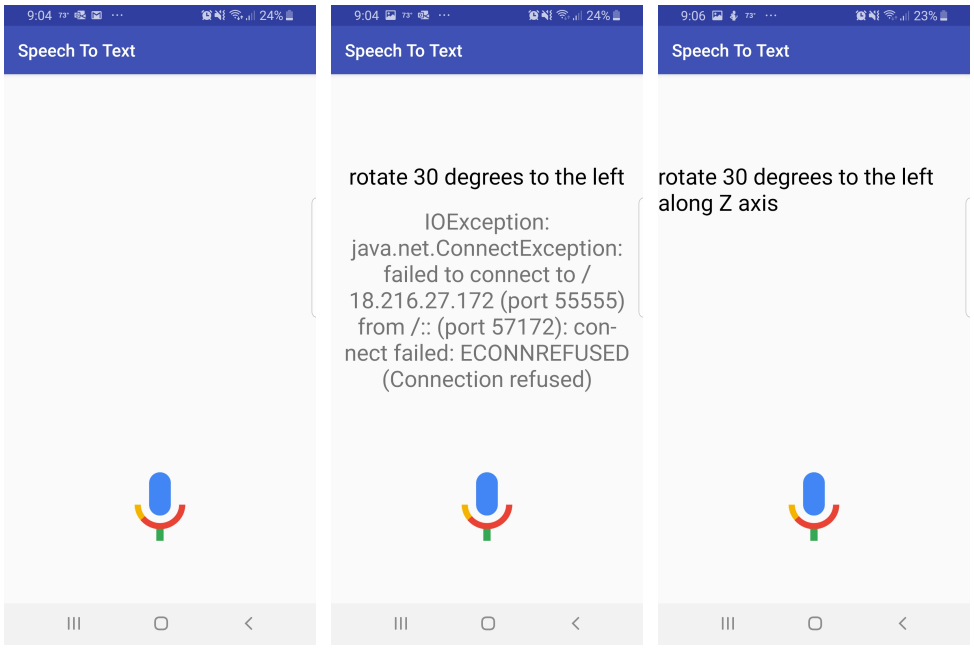
Figure 3.1: System Architecture

3.1 Speech Navigation Application (SNA)

The Speech Navigation Application (SNA) is designed and implemented using Android Studio. I have used Android8.0 (Oreo) version for the SNA. Its main functionality involves receiving the user's commands in the form of speech. The spoken commands are then converted to text using the Google Cloud Speech-to-text API. The Main layout of the application is as shown in Figure 3.2(a) which represents the overall application layout.

From Figure 3.2(a), it can be observed that the User Interface's(UI) contains a mic button that when pressed on allows the user to record the message or command that they wish to see reciprocated on the screen. The button on being pressed, waits till the user finishes speaking and automatically stops recording. On finishing the recording, the Google Cloud Speech-to-Text API converts the spoken content into text. These speech to text converted content or messages are displayed in the text area of the application.

The second Figure 3.2(b) represents the application display on receiving an exception. The application is programmed to throw exceptions in case it cannot connect to the server



(a) Application Layout (b) On Exception (c) Command Recieved

Figure 3.2: Application Layouts

with the help of a try catch block. The layout of the application remains the same, but the text area displays the exception message. This exception message is generated when the application cannot connect to the server as it is either busy or not running.

The third Figures 3.2(c) represents the application display, when the command is recieved and has been successfully sent across to the server. This display represents the successful transfer of the speech to text converted message to the server. The Speech Navigation Application communicates with the server with the help of Java socket programming. The type of communication that is being used here is TCP (transmission control Protocol). The flow chart represented by Figure 3.3 describes the overall working of the Application.

The work flow of the application starts with the user clicking on the mic button. On clicking the mic button, the user message is recorded. This recorded message is then converted into text after it gets processed by Google Cloud Speech-to-Text API. This converted message is then displayed in the text area in the middle of the application.

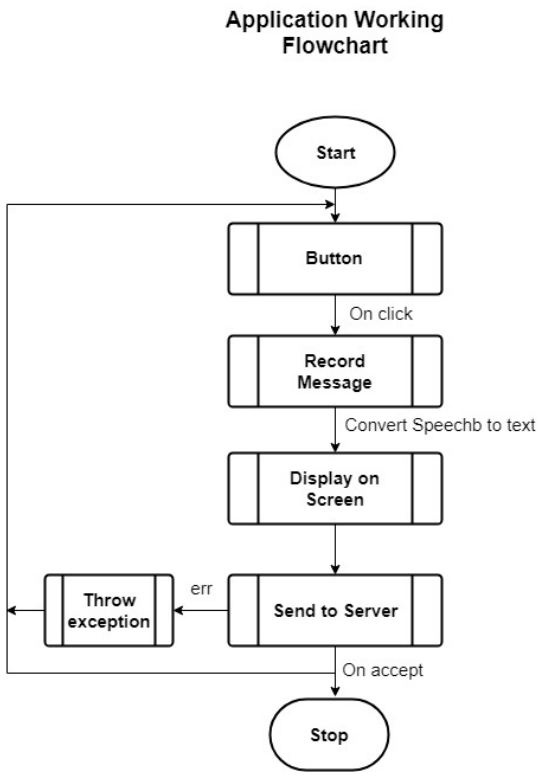


Figure 3.3: Application Working

On displaying the speech converted text, the same text which is stored in a variable, is sent to the server with the help of java socket programming.

If the communication could not be established with the server, an exception message is thrown by the try- catch block and the process starts all over again. However the users message will be stored in a queue, on re-establishing the communication with the server, the message will be sent across. The entire message in the form of text is sent to the server to be processed into commands termed as the command string (CS). The processing of the command string and the various situations that are encountered while receiving or not receiving a command from the application will be explained in more detail in the following sections. The main function of the SNA in terms of networking, is that it start with the initial handshake with the server to establish communication, and then keeps sending the users input as and when it is available. Google's Cloud Speech-to-Text API plays a major role in the applications functionality. The default language that the speech-to-text API uses

is English. But it does support a variety of other languages, 120 languages to be precise [1]. Adding more languages is part of the future implementation of the SNS. This way the SNS can be used by users who are not from different lingual backgrounds.

3.1.1 Speech Navigation Application Program Summary

Android studio's uses the drag and drop method of programming, where each layout can be designed using the available predefined widgets it supports. For each layout designed, the corresponding XML program and java class is created. The XML code can be used to alter the look of the widgets. Where as the Java class can be used to add additional functionality to the layout.

The Speech Navigation Application(SNA) has a single main layout. In the layout I have included a text area and a button, using the drag and drop option. This generated a Main Activity Java class and the corresponding Main Activity XML program.

In the Main Activity Java class I incorporated the Google Cloud Speech API, by mentioning the respective intents. The Main Activity Java class has a function to get the speech converted to text input called the getInput() function. This function, takes the users spoken contents, and converts the spoken content to text using Google Cloud Speech-to-Text API. The default language setting is English. The result is then set in the text area using the setText function.

On displaying the result in the text area, the execute method in the MessageSend Class is called to send the speech converted text to the server. The MessageSend Class has a java socket setup, by first connecting to my server IP address in a try catch block. If the socket could not establish communication with the Server an exception is thrown and the exception message is set in the Text area of the Main Layout. If the server establishes communication with the server, it sends the Speech converted to Text message to the server.

Here,the main function of the Java socket is to write to the server after the initial handshake to establish communication has been made. The type of communication established

here is Transmission Control Protocol (TCP).

3.2 Server Implementation

The Server Side is implemented using Amazon Web Services (AWS). I used the Elastic Compute Cloud (EC2) Instance to design and implement my Server. I used the Ubuntu free tier EC2 instance. The Server is completely programmed in python, and uses python socket programming. Once again using the Transmission Control Protocol (TCP) to establish communication. The server-side communication is divided into 2 main parts namely Application to server communication and Server to Display side Client communication. Separate ports were used to establish communication between the application with Server and Server with Display Side Client. This was done to manage the traffic in a more efficient way. The following Figure 3.4 flowchart represents the overall working of the Server. The server starts by first establishing communication with its corresponding ports. On connecting to the ports, it waits for the Speech-to-text message(STT) from the application. On receiving the STT request message, it processes the STT message into specific commands called the Command String(CS). This CS is then sent to the Display Client to process the command further.

3.2.1 App to server communication

The server is programmed to receive the spoken commands in the form of text. The received text contains keywords that form the main commands that will be sent to the client side. Before the text is received the server establishes communication with the Application through a TCP handshake process. The server is then ready to receive the STT message from the application. On receiving the STT message, the text is stored in a string. This string is then processed to look for specific words that help form the Command String(CS).

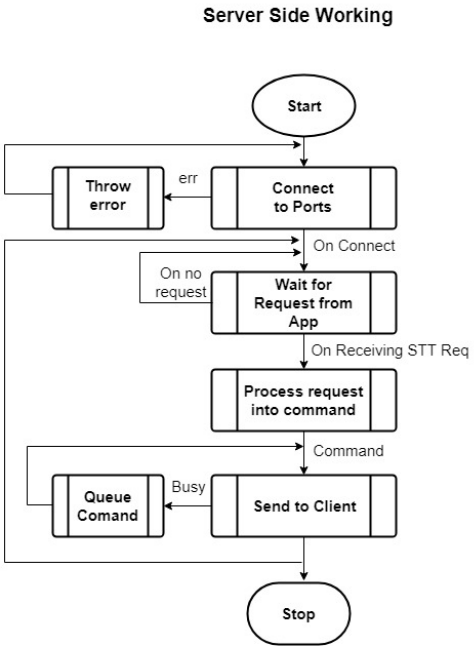


Figure 3.4: Server Implementation Flowchart

The key words are all read and ordered into the command format. The various commands supported by SNS will be discussed in a separate section later in the paper.

3.2.2 Server to client communication

Like the App to Server communication, before any transaction takes place between the server and the client side, a communication is established with the initial handshake. The CS is created based on the input received in the form of text from the application, is sent to the client side for further processing. This transaction takes place through a different port, for a more secure connection. The CS is then sent to the client side for further processing.

3.3 Command Processing

The Server receives the STT message from the application. This STT Req is the raw conversion of the spoken commands, voiced out by the user in the form of text. This

STT message is stored in a string as mentioned earlier. The string will then be processed using the Python string find() method. The find method processes the string to look for key words. The presence of these key words determine the formation of the Command String(CS).

The good thing about this method of command processing is that the user can state the command however they feel is correct. The commands however, will only execute when all the key words necessary for the processing is voiced out. For example, lets consider that we want to perform the rotation command on the visualization. In order to execute the rotation command, the system requires you mention 4 main key words. The first and most important keyword is the word ”Rotate”. The second keyword required is the direction in which you want to rotate the visualization. The SNS supports ”clockwise”, ”anticlockwise”,”left” and ”right” keywords responsible for the direction in which you wish to rotate the visualization respectively. The third keyword that is important is the axis along which you would like for the visualization to rotate. The keywords can be either ”x-axis”, ”y-axis” or ”z-axis”. The fourth and final keyword is the degree by which you would like for the visualization to rotate. Therefore the total sentence formation can be voiced out as ”Rotate 30 degrees to the left along y-axis” or ”Rotate anticlockwise by 30 degrees along y-axis” or ”Rotate 30 degrees along y-axis to the left”. All these strings perform the same function despite the order in which they are said. They will all rotate the Visualization anticlockwise, by 30 degrees along the y-axis. The commands supported by the SNS are of two types, namely View Setting commands and Algorithm based commands. The following subsections will give you an overview of all the commands and the keywords that are needed to be mentioned in order for the respective commands to execute.

3.3.1 Python Server Program Summary

The Python program layout for the server is similar to the working represented by the flowchart in Figure 3.4. The program flow starts with initializing and binding the sockets

to the corresponding ports. Here I have assigned two separate ports to handle the application and server communication and the server and the display client communication. The sockets follow the TCP protocol.These sockets after being initialized and bound to the respective ports, are then set to listen to the corresponding ports for any message requests. The port assigned to listen to the application is 55555 and the port assigned to listen to the Display client is 55556.

The port 55555 reads the input from the application. Here I have set a python socket timeout to 0.1 seconds. Here if the port does not return a message within the timeout range, a default message string ”do nothing” is set as the command string(CS), to be sent across to the Display Client. If any messages are read from port 55555, that message is then processed to derive the command string.

The process of deriving the command string is different for each command. A series of if-else loops are used to find the corresponding keywords that are needed to execute the respective commands. Here the inbuilt find function in python programming is used to traverse through the incoming string from port 55555, to find each particular keyword. Only if all the conditions are satisfied(if all the keywords to complete a command are found), the command string is formed and is sent to the display client.

Here, the socket assigned to the Display Client(DC) only sends the command string messages to the DC using port 55556.The whole process starting from the part where we read from port 55555 till writing to port 55556, is placed inside a while loop so that the same process repeats until the server is terminated manually.

3.4 View setting Commands

The Speech Navigation System supports the following view setting commands. These commands deal with rotating, scaling and panning the visualization to the desired angle, axis and amount. These commands are handled by the Speech Navigation Tool code.

3.4.1 Command format for rotation (R AN X Degree)

Here R stands for Rotation, the second term determines the direction of rotation and can be AN(Anticlockwise) and CL(Clockwise). The third term determines the axis along which you wish to rotate the visualization. The fourth and final term represents the degree with which you wish to rotate the visualization.

3.4.2 Command format for Panning (P L Val)

The following Syntax is used to represent the Command for Panning:

Here the first term represents Panning. The second term represents the direction in which you wish to rotate the visualization, which can either be (L) left, (R) right, (U) up or (D) down . The third term represents the value with which you wish to translate the visualization in the direction you wish.

3.4.3 Command format for Scaling (S ZIN 30)

Here the first term is used to distinguish the command as scaling. The second term represents whether you wish to zoom in(ZIN) or zoom out(ZOUT). The last term represents the value by which you wish to zoom in or zoom out the visualization.

3.5 Algorithm and Parameter based commands

The Speech Navigation System supports the following commands to change algorithms and the parameters in them. This functionality is handled by the main program. Each algorithm based command is distinguished by the first term represented by ALG.

3.5.1 Command format for changing algorithms (ALG Algorithm-name)

In this command the first term represents the type of command to be processed, ie algorithms. The second term represents the name of the algorithm that you wish to run. This command will be activated when you mention the name of the algorithm followed by the term ”algorithm” when using the app. The Speech Navigation system at present supports 4 algorithms, namely sphere, Cone, Cylinder and Cube.

3.5.2 Command format for adding another actor to the visualization (ALG ADD Algorithm-name)

This command works similar to the previous command, but it is used to add an additional algorithm (aka Actor), to the existing visualization. The second term ADD, helps distinguish the previous command by representing the addition of another actor.

3.5.3 Command format for changing color (ALG COL Color)

I this format the first term as all algorithm parameters are represented by ”ALG”. The second term represents the parameter that we wish to change, ie color. The third term represents the color we wish to change the visualization to . The third supports 4 colors at present and they are Red, Green, Blue and Yellow. Each of those colors are abbreviated to R, G, B and Y respectively.

3.5.4 Command format for changing radius (ALG RAD Val)

This command can be used to change the radius of the algorithm that support radius functionality. At present this parameter can alter the radius of the sphere algorithm, cylinder algorithm and the cone algorithm.

3.6 Client Side Implementation

The client side is designed and implemented using VRUI and VTK. It is programmed using C++ and C++ socket programming.The client side implementation involves 3 main sets of code/programs. They are the Main Code, Speech Navigation Tool(SNT) code and the Server Communication Code. Each of these codes handle different aspects of the clients functionality. Each will be explained in detail in the following sections. The final processed visualization as a result of the command processing is then displayed on the DIVE environment [15].

The client side is comprised of 3 parts. The first part is the main code where the VTK code runs and the respective VTK algorithms are executed. The second part is the Speech navigation Tool Code, it handles all the changes to be made to the actor's orientation. The third part is the server communication code, which reads from the server to retrieve the CS. Figure 3.5 represents the Display Client working. The Display Client, on establishing a secured connection with the server, receives the abbreviated form of the command also known as the command string (CS). The CS is then carefully read for abbreviations that specify what variables need to be changed. On reading the input, if it receives commands for basic movement of the visualization such as rotate, scale and pan, the corresponding variables responsible for the respective visualization are changed. However, on receiving the command to alter the algorithm, the commands are handled by the main code. Based on the algorithm that the user wishes to see reciprocated on the screen, respective changes are made to display it.

3.6.1 Server Communication Code(SCC)

The incoming commands are read by the server communication code(SCC). The SCC is implemented using C++ programming. Its main function is to handle the incoming data from the server. It is designed such that it will only read from the server using C++ Sockets

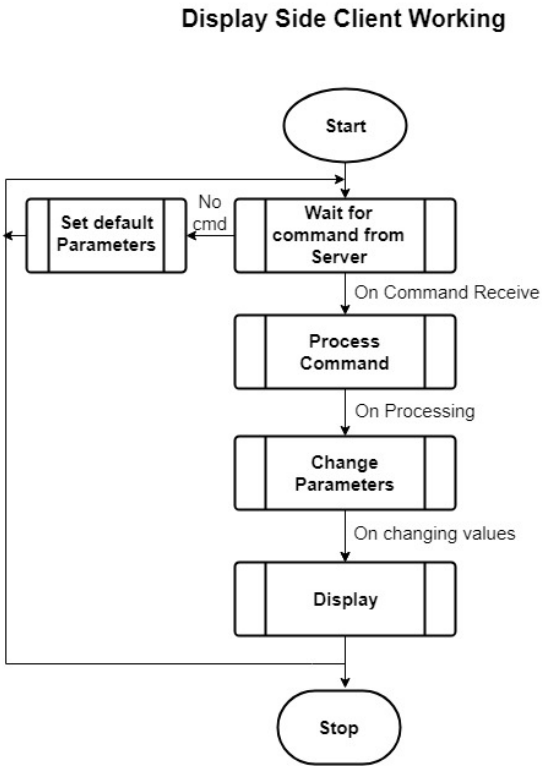


Figure 3.5: Display Side Client Flowchart

and return any acknowledgment message to the server in return. Here the data from the SCC is read into a string variable called data. This data is then sent across to both the Speech Navigation Tool code as well as the Main code.

3.6.2 The Main Code

The Main Code(MC) is designed using VTK and VRUI. Here the algorithms that we wish to see displayed are incorporated. The VRUI frame work handles the VTK algorithms and displays the visualization on the corresponding VR displays. The algorithms are all compiled in the Setup() function where the VTK pipeline is Rendered. The Pipeline is formed in the ProcessList(PL) function. As the name suggest the PL function processes the VTK pipeline that the user wishes to visualize in the display. In MC there is a main list called the algorithm list(AL). The Algorithm list stores the algorithms that the user

wishes to see displayed. Each algorithm according to the VTK pipeline, conatins a Source, mapper, actor and Renderer.

The Main Code reads the incoming data from the server from SCC into the frame function. The frame function executes around 60 times per second. So the server is constantly being read. If there is nothing to be read from the server, the default string ”do noting” is assigned to the data string. If there is something to be read from the server, the data received will be the command string(CS) sent form the server. On reading the data the Process List function is called. Here the CS is processed using the built in c++ find operation to search the CS to see if it has an algorithm based command. The CS is termed to be an algorithm based command if the keyword ”ALG” is present. If the ”ALG” keyword is not present in the CS, the MC will not process the CS as it terms the CS to be a non-Algorithm Based command. If the ”ALG” keyword is found, the PL function processes the command and calls the Setup function. The PL function handles both the Algorithm pipeline setup, in the form of a list, as well as commands to make corresponding changes to the algorithm parameters according to the changes mentioned in the CS.

If the CS contains the command to change the algorithm, for example ”ALG CYL”, where ”CYL” represents the short form of the Cylinder Algorithm. The PL function adds the corresponding source for the Cyliner algorithm, the Cylinder Mapper, the Cylinder Actor and the Renderer to a list called the sub list. This sub list is then pushed into the main algorithm List. The PL function after processing this list calls the Setup function.

If the CS contains the command to change the algorithm parameter, the PL function processes the CS to change the corresponding parameter values by assigning the parameter values to the desired values in the current algorithm that is running. On changing the parameter values, the PL function calls the setup function.

Here, the Setup function, runs the nested list to display the algorithm that is mentioned in the CS to be visualized or updates the current algorithm with the changed parameter values as per the CS. The List structure mentioned in the PL funciton, supports any algorithm,

as it stores the main pointers i.e. the Source, Mapper, Actor and the Renderer in it. Each VTK pipelines or algorithms includes all these pointers. Hence making the SNS more flexible in terms of supporting most of the algorithm based libraries in VTK.

3.6.3 Speech Navigation Tool Code (SNT)

The Speech Navigation Tool Code deals mainly with the VRUI framework which manipulates the View Settings. Its layout is similar to the Main Codes layout. The incoming commands are read into the Frame function, from server communication code. Based on the type of command that is read, if the command is based on changes that are needed to be made to the view setting, that command will be processed by the Speech Navigation Tool(SNT) code.

The view setting commands are comprised of scaling panning and rotation commands. On receiving the view setting commands, corresponding changes are made to the respective variables in the VRUI framework in order to alter the visualization.

Results and Analysis

The Speech Navigation System was run to take user commands and the corresponding images represent the changes made to the visualization, based on the command given. I have considered an simple Cylinder algorithm to show the working of the View Setting commands. VTK supports a vast library of algorithms. I have used the VTK Object-Reader algorithm to visualize a dragonfly in the Algorithm changing command. As there are quite a few commands that can be performed by the SNS, I have given one example of each type of command that it supports.

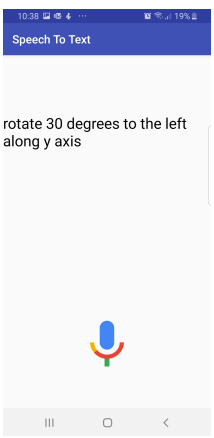
4.0.1 View Setting Command Results

The following Images represent the working of the SNS for the corresponding view setting commands it supports. I have given an example for each type of commands that the SNS supports. The main types of View Setting commands supported by the SNS are Rotation, Scaling and Panning. All images of the visualizations represented, are taken in the DIVE environment.

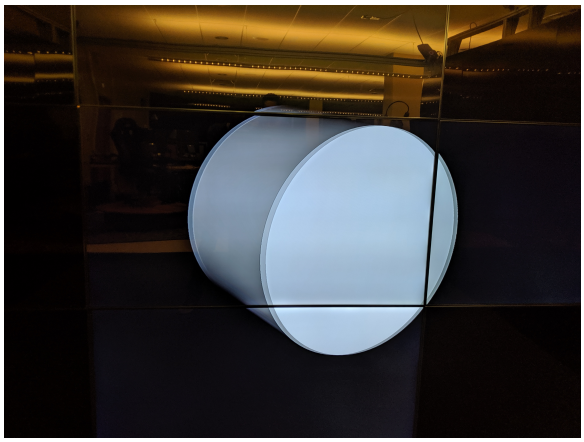
1. **Rotation Command:**

   The Figure 4.1(a) represents the command voiced out by the user to rotate the visualization. Here, the user has quoted to ”rotate the visualization by 30 degrees to the left along y axis”. The corresponding Image 4.1(b) represents the updated visualization after the command has been processed. The Image represents the cylinder algorithm,

rotated by 30 degrees, counter-clockwise along the Y-axis.



(a) Rotation command

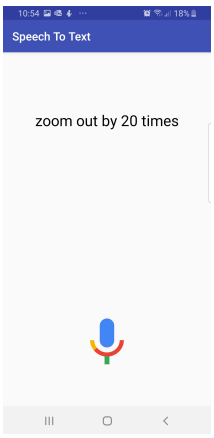


(b) On Rotation

Figure 4.1: Rotating

2. **Scaling Command:**

Figure 4.2(a) represents the command voiced out by the user to scale the visualization. Here, the user has quoted ”Zoom out by 20 times”. The Figure 4.2(b) represents the updated visualization after the command has been processed. The visualization of the cylinder is zoomed out by 20 times from its initial representation.



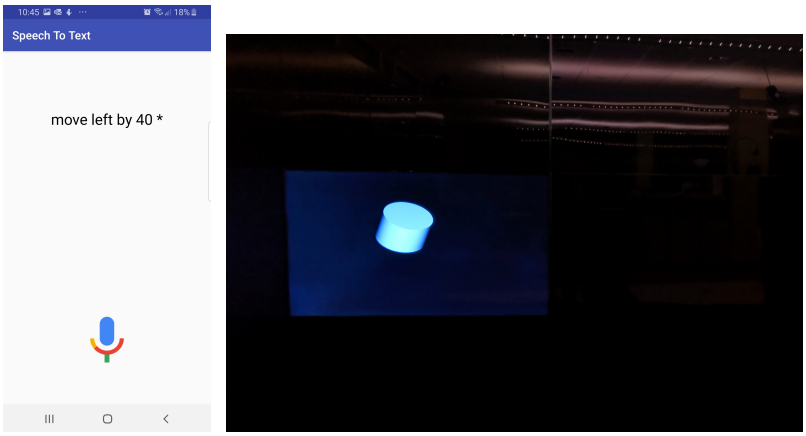
(a) Scaling command

(b) Scaling Display

Figure 4.2: Scaling

3. **Panning Command:**

   Figure 4.3(a) represents the command voiced out by the user to perform panning operation on the visualization. Figure 4.3(b) represents the updated visualization after the command has been processed. Here, the visualization is panned to the left by 40 times. This is the image obtained after the scaling command.



(a) Panning command (b) Panning Display

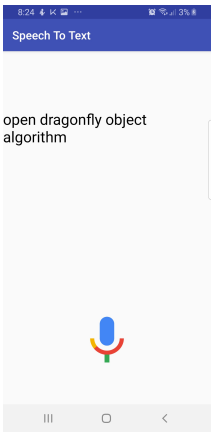
Figure 4.3: Panning

4.0.2 Algorithm command results

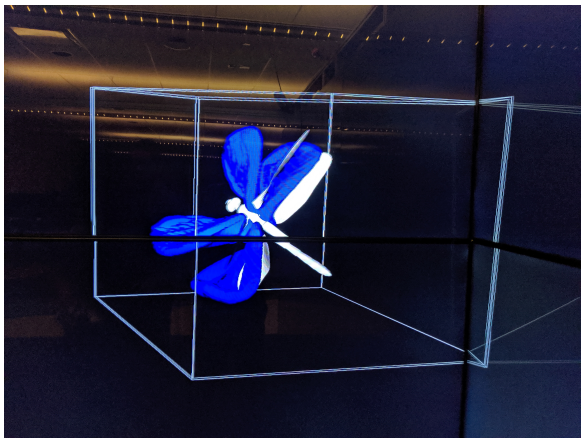
The following diagrams represent the corresponding changes to execute algorithms and the parameters in the algorithms.

1. **Change Algorithm:**

   The Figure 4.4(a) represents the command voiced out by the user to change the algorithm to be visualized, by mentioning the algorithm that they would like to see on the display. In this image the user has quoted ”open dragonfly object algorithm”. The Figure 4.4(b) represents the updated visualization after the command has been processed and represents the dragonfly object file after it has been processed using the vtkObjectReader library.



(a) Algorithm Command

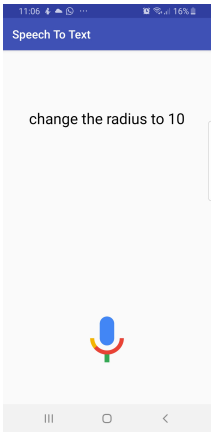


(b) Dragonfly data set

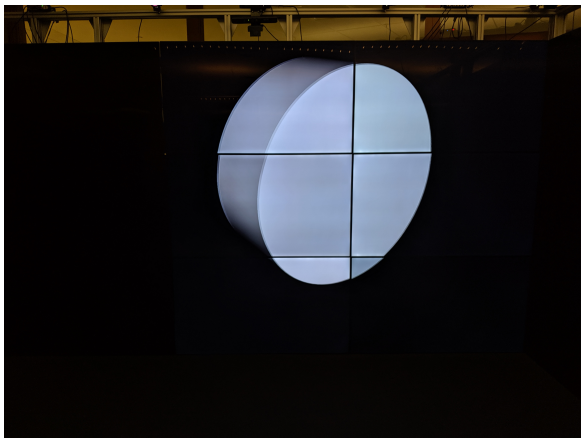
Figure 4.4: Changing Algorithms

2. **Changing parameters in Algorithms:**

   Figure 4.5(a) represents the command voiced out by the user to change the parameter in the algorithm, to be visualized. The user is quoted to be asking to ”change the radius to 10”. here I have used the cylinder algorithm as an example. Figure 4.5(b) represents the updated visualization after the command has been processed, i.e. the radius of the cylinder has been changed from a radius of 5, to a radius of 10, on processing the command.



(a) Parameter Command



(b) Changed Radius

Figure 4.5: Changing Algorithm Parameters

4.0.3 User feedback

A number of participants were asked to provide feedback on their experience with the SNA and the overall working of the system. One of the questions that the participants was asked was whether they could see the changes in the display as per their requirements. Figure 4.6 represents the statistics of the participant responses.

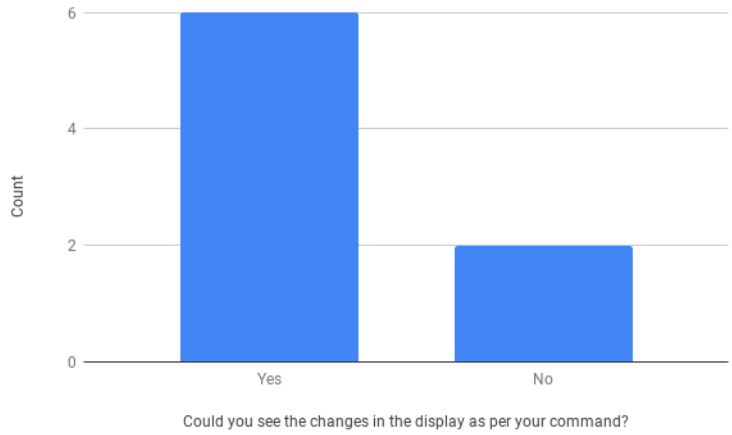


Figure 4.6: Changes as per commands

The Participants were then asked if any of their commands were miss interpreted and hence the wrong command was executed. Figure 4.7 represents the statistics of the responses that the participants provided based on their experience with the SNA.

The Participants were also asked to provide their input on their experience with the Speech Navigation Tool, and to express their ideas on improving the system. The following participant inputs seemed to be a good addition to the overall system.

1. To add more commands that use words like turn in place of Rotate.

2. To introduce a small time gap before the app stops recording or implement the button on the app to stop recording after the entire command is voiced out,by pressing it again especially if the user had not completed voicing out the command.

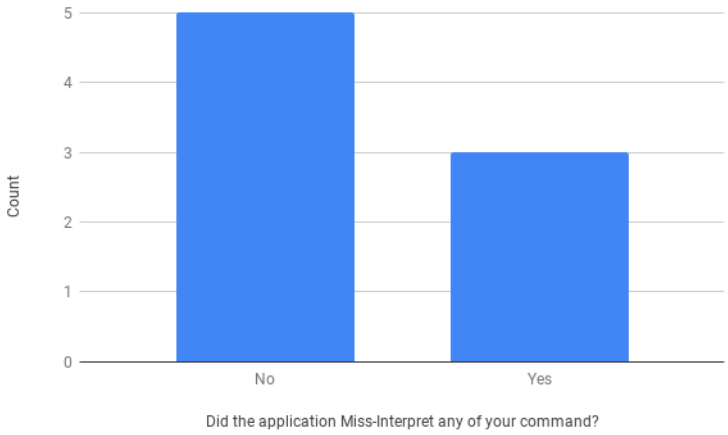


Figure 4.7: Speech Recognition Rating

3. Giving a name to the device, similar to hey google or Jarvis. As a lot of users may be familiar with virtual assistants like Amazon Alexa or the Google Assistant.

Conclusion

Based on the evaluation of the working of the overall system, the Speech Navigation Application fared pretty well in terms of processing the commands. There were a few issues with mispronunciation, and the app misinterpreting the commands, but the SNS executed the users commands and the corresponding changes to the visualization could be seen on the DIVE environment.

5.0.1 Other Possible Applications of The Speech Navigation System

The Speech Navigation System can be used in multiple fields. Some of the possible uses of the Speech Navigation system are as follows:

1. For visualizing large data-sets, as VTK supports the representation of point data in a 3D space. It can be used to perform various machine Learning Algorithms.

2. It can be used in Virtual Shopping environments to navigate through the isles. The application can house additional features like a cart for adding the items to be purchased.

5.0.2 Future Work

The Speech Navigation system is pretty flexible in terms of accommodating additional commands as well as algorithms. The list structure in the Display Client, Main program,

provides the flexibility to adding additional algorithms. However, there is still scope for improvement. The following ideas are in consideration for future implementations.

1. Accommodate More Algorithms for more complex visualizations. This is possible due to the large number of algorithms supported by VTK.

2. Provide algorithm specific parameter options. Where the SNA can display the commands supported by the algorithm that is being visualized. This way, the user will know which commands can be performed on that particular algorithm.

3. Another possible scope for the SNS, is to get the commands working in multiple different languages. This way users from different lingual backgrounds can work with the SNS.

Bibliography

[1] Google Cloud Speech-to-Text official site. https://cloud.google.com/speech-to-text/.

[2] Douglas A Bowman and Larry F Hodges. User interface constraints for immersive virtual environment applications. Technical report, Georgia Institute of Technology, 1995.

[3] Phuc Ky Do and Justin Monroe Pierce. Virtual shopping environment, February 20 2007. US Patent 7,178,722.

[4] Víctor Manuel Álvarez García, María del Puerto Paule Ruiz, and Juan Ramón Pérez Pérez. Voice interactive classroom, a service-oriented software architecture for speech-enabled learning. *Journal of network and computer applications*, 33(5):603–610, 2010.

[5] Stuart Goose, Sandra Sudarsky, Xiang Zhang, and Nassir Navab. Speech-enabled augmented reality supporting mobile industrial maintenance. *IEEE Pervasive Computing*, 2(1):65–70, 2003.

[6] Veton Këpuska and Gamal Bohouta. Comparing speech recognition systems (microsoft api, google api and cmu sphinx). *Int. J. Eng. Res. Appl*, 7(03):20–24, 2017.

[7] Arjan JF Kok and Robert Van Liere. A multimodal virtual reality interface for 3d interaction with vtk. *Knowledge and Information Systems*, 13(2):197–219, 2007.

[8] Oliver Kreylos. Environment-independent vr development. In *International Symposium on Visual Computing*, pages 901–912. Springer, 2008.

[9] Jianfeng Lu, Zhigeng Pan, Hai Lin, Mingmin Zhang, and Jiaoying Shi. Virtual learning environment for medical education based on vrml and vtk. *Computers & Graphics*, 29(2):283–288, 2005.

[10] Siddhesh Manjrekar, Shubhrika Sandilya, Deesha Bhosale, Sravanthi Kanchi, Adwait Pitkar, and Mayur Gondhalekar. Cave: An emerging immersive technology–a review. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 131–136. IEEE, 2014.

[11] W Dean McCarty, Steven Sheasby, Philip Amburn, Martin R Stytz, and Chip Switzer. A virtual cockpit for a distributed interactive simulation. *IEEE Computer Graphics and Applications*, 14(1):49–54, 1994.

[12] William J Schroeder, Lisa Sobierajski Avila, and William Hoffman. Visualizing with vtk: a tutorial. *IEEE Computer graphics and applications*, 20(5):20–27, 2000.

[13] MM Sharma. *Starting with Android*. BPB Publications, 2018.

[14] Lai Khin Wee, Hum Yan Chai, and Eko Supriyanto. Surface rendering of three dimensional ultrasound images using vtk. 2011.

[15] Thomas Wischgoll. Display systems for visualization and simulation in virtual environments. *Electronic Imaging*, 2017(1):78–88, 2017.

[16] Thomas Wischgoll, Madison Glines, Tyler Whitlock, Bradley R Guthrie, Corinne M Mowrey, Pratik J Parikh, and John Flach. Display infrastructure for virtual environments. *Electronic Imaging*, 2018(1):1–11, 2018.

[17] W Zhao and V Madhavan. Integration of voice commands into a virtual reality environment for assembly design. In *Proceedings of the 10th annual international conference on industrial engineering theory, applications & practice, Clearwater Beach, FL, USA*, 2005.